

NASA TECHNICAL MEMORANDUM

NASA TM X-64878

SYSTEM SUPPORT SOFTWARE FOR THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC)

By T. E. Hill, G. C. Hintze, B. C. Hodges
Data Systems Laboratory

F. A. Austin, B. P. Buckles, R. T. Curran,
J. D. Lackey, R. E. Payne
Computer Sciences Corporation

September 23, 1974



NASA

*George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama*

(NASA-TM-X-64878) SYSTEM SUPPORT SOFTWARE
FOR THE SPACE ULTRARELIABLE MODULAR
COMPUTER (SUMC) (NASA) 41 p HC \$3.25

N74-33677

CSCL C9B

Unclass

G3/08 49310

1. REPORT NO. NASA TM X-64878	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.	
4. TITLE AND SUBTITLE SYSTEM SUPPORT SOFTWARE FOR THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC)		5. REPORT DATE September 23, 1974	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) T. E. Hill, G. C. Hintze, B. C. Hodges, F. A. Austin*, B. P. Buckles*, R. T. Curran*, J. D. Lackey*, R. E. Payne*		8. PERFORMING ORGANIZATION REPORT #	
9. PERFORMING ORGANIZATION NAME AND ADDRESS George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812		10. WORK UNIT NO.	11. CONTRACT OR GRANT NO.
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D. C. 20546		13. TYPE OF REPORT & PERIOD COVERED Technical Memorandum	
15. SUPPLEMENTARY NOTES * Computer Sciences Corporation		14. SPONSORING AGENCY CODE	
16. ABSTRACT <p>This report describes the highly transportable programming system designed and implemented to support the development of software for the Space Ultrareliable Modular Computer (SUMC). The SUMC System Support Software (S4) consists of program modules called processors.</p> <p>The initial set of S4 processors consists of the S4 Supervisor, the General Purpose Assembler for SUMC instruction and microcode input, Linkage Editors, an Instruction Level Simulator, a Microcode Grid Print Processor, and user oriented utility programs. A FORTRAN IV compiler is undergoing development. The design of S4 facilitates the addition of new processors with a minimum effort. S4 provides the user quasi host independence on the ground based operational software development computer. S4 provides the additional capability to accommodate variations in the SUMC architecture without consequent major modifications in the initial S4 processors.</p>			
17. KEY WORDS		18. DISTRIBUTION STATEMENT Unclassified-unlimited	
19. SECURITY CLASSIF. (of this report) Unclassified	20. SECURITY CLASSIF. (of this page) Unclassified	21. NO. OF PAGES 41	22. PRICE NTIS

TABLE OF CONTENTS

		<u>Page</u>
SECTION I.	INTRODUCTION.	2
SECTION II.	SUMC SUPPORT SOFTWARE SYSTEM (S4) DESIGN CONCEPTS	3
SECTION III.	SOFTWARE DESCRIPTION	6
	A. General.	6
	B. S4 Supervisor.	9
	C. S4 FORTRAN Cross Compiler.	15
	D. S4 Cross Assemblers	22
	E. S4 Linkage Editors	26
	F. S4 Instruction Simulator.	29
	G. S4 Microcode Grid Print	33
SECTION IV.	CONCLUSIONS AND RECOMMENDATIONS. .	35

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1	Initial Set of S4 Processors	7
2	S4 System Processor Control and Data Flows .	8
3	S4 Supervisor Overlay Control	13
4	SUMC FORTRAN IV Compiler Dependency Chart.	16
5	Basic Program Modules for SUMC Instruction Simulator	30

PRECEDING PAGE BLANK NOT FILMED

UNUSUAL TERMS

Processor - A major S4 program entity designed to accomplish a specific software task. A processor may be added to S4 without major impact on the system.

General Purpose Assembler - An S4 processor constructed to accept a set of SUMC target computer design definition parameters with corollary SUMC mnemonic commands. These inputs are translated by the General Purpose Assembler into an intermediate level output which is processed to produce SUMC object code.

NONSTANDARD ABBREVIATIONS

IL	Intermediate Language
S4	SUMC Support Software System
SUMC	Space Ultrareliable Modular Computer
VSD	Vector Symbol Dictionary

TECHNICAL MEMORANDUM X-64878

SYSTEM SUPPORT SOFTWARE FOR THE SPACE ULTRARELIABLE MODULAR COMPUTER (SUMC)

SUMMARY

The Space Ultrareliable Modular Computer (SUMC) Software Support System was designed and implemented to support the SUMC operational hardware. This support software system is written primarily in Standard FORTRAN IV and is highly portable from one host computer system to another.

The software is constructed in program units called processors. The controller processor, called the S4 Supervisor, communicates with the host computer operating system and manages the execution of subordinate S4 processors such as assemblers, compilers, and linkage editors. Host computer dependencies, such as word size, are coded as parameters in the S4 Supervisor data tables. Processors executing under control of the S4 Supervisor utilize these tables to mask host computer dependencies. This concentration of the host computer design parameters in the S4 Supervisor data tables facilitates modification in the transfer to a different host computer system. The remaining host dependencies are masked by utilization of a few subroutines written in host computer assembly language. S4 minimizes computer host dependence and provides the SUMC software programmers the tools to develop flight software for variable SUMC hardware architectures.

SECTION I. INTRODUCTION

The S4 design concepts arose from efforts to provide a host computer independent software system containing the processors necessary for generation of software for the Space Ultrareliable Modular Computer (SUMC).

The SUMC family is an emerging class of spaceborne computers intended for missions during the 1974-1980 time frame. SUMC design concepts emphasize four-bit incremental word length, microprogrammable instruction sets and expandable scratch pad and main memories. Additionally, multiprocessor organizations are envisioned utilizing the SUMC as the repeated computer module.⁽¹⁾ As a self-contained software system, S4 supports the development of programs for the SUMC family of computers. Reduction of software support development programming effort is accomplished through the following S4 design goals:

- o Maximizing S4 host computer independence and thereby minimizing the effort required to transport S4 from one host computer to a different host computer.
- o Defining different SUMC target computers via parameterization.

SECTION II. SUMC SUPPORT SOFTWARE SYSTEM (S4) DESIGN CONCEPTS

S4 is designed to accelerate SUMC support software development on a variety of host computers. The host transportability and capability are accomplished by the following:

- o S4 is written primarily in ANSI FORTRAN IV with host dependent programming isolated in the supervisor. (2, 3)
- o Host dependent parameters are input as system variables and utilized to control processor execution in the current host environment.
- o S4 is constructed in a modular, open-ended form with the supervisor controlling execution of processors such as compilers, assemblers, linkage editors, and simulators.
- o User file construction and maintenance are defined and implemented for interfacing with the host I/O.

For S4 implementation a minimum host hardware configuration is required as shown in Table I, Host Computer Configuration Requirements. Most existing medium and large scale commercial and scientific computer installations easily exceed the noted configuration requirements.

In addition to the host hardware requirements, certain host operating system capabilities are required:

- o An overlay mechanism to load and execute user programs.
- o Basic I/O handlers to support hardware requirements.
- o Disc file management with random disc addressability of declared contiguous disc space.

TABLE 1. HOST COMPUTER CONFIGURATION REQUIREMENTS

HOST COMPUTER CONFIGURATION	SIZE PARAMETERS
CPU	1
MAIN STORAGE (User available)	64K WORDS
PRIMARY INPUT	CARD READER
PRIMARY OUTPUT *	CARD PUNCH
MAGNETIC TAPE UNITS	3 LOGICAL UNITS
DISK	APPLICATION DEPENDENT ($\approx 1.0 \times 10^7$ WORDS)

*PAPER TAPE PUNCH MAY BE UTILIZED IF REQUIRED BY THE TARGET COMPUTER

In the following sections, the S4 functional design will be presented. The initial set of S4 processors is described in the Software Description (Section III). Ensuing sections of this report are devoted to a description of the design of the processors that comprise the initial S4:

- o S4 Supervisor.
- o FORTRAN Cross Compiler.
- o Cross Assemblers.
- o Linkage Editors.
- o Simulator.
- o Microcode Grid Print.

SECTION III. SOFTWARE DESCRIPTION

A. General

The S4 processors are constructed as an open-ended collection and designed to provide the user with the capability to generate SUMC software. The initial S4 system, noted in Figure 1, is made up of the following software support processors:

- o S4 Supervisor.
- o FORTRAN Cross Compiler.
- o Cross Assemblers.
 - Instruction Assembler
 - Microcode Assembler
- o Linkage Editors.
 - Instruction Linkage Editor
 - Microcode Linkage Editor
- o Simulator.
 - Instruction Level Simulator
- o Target Output Converters.
 - Instruction Load Module Converters
 - Microcode Load Module Converters
- o Microcode Grid Print

Figure 2 describes the interrelationship of the S4 processors. It demonstrates the possible processor control and data flows necessary to generate SUMC target load modules. Choice(s) among the possible paths is controlled by the supervisor via the S4 control language (Section III, B. 2). Source modules can be translated and/or saved in a user external file. Compiler output is in a format acceptable to the

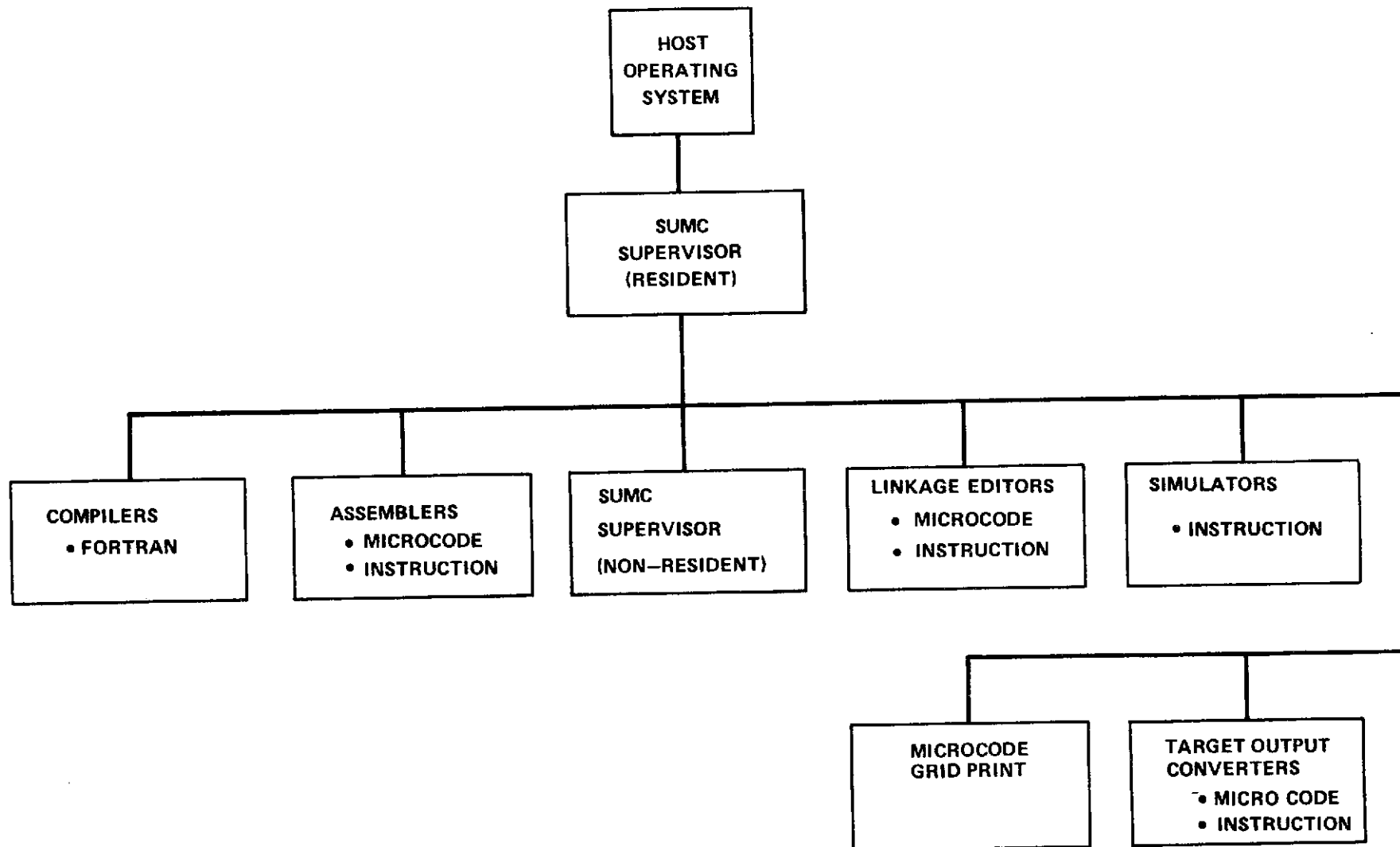


FIGURE 1. INITIAL SET OF S4 PROCESSORS

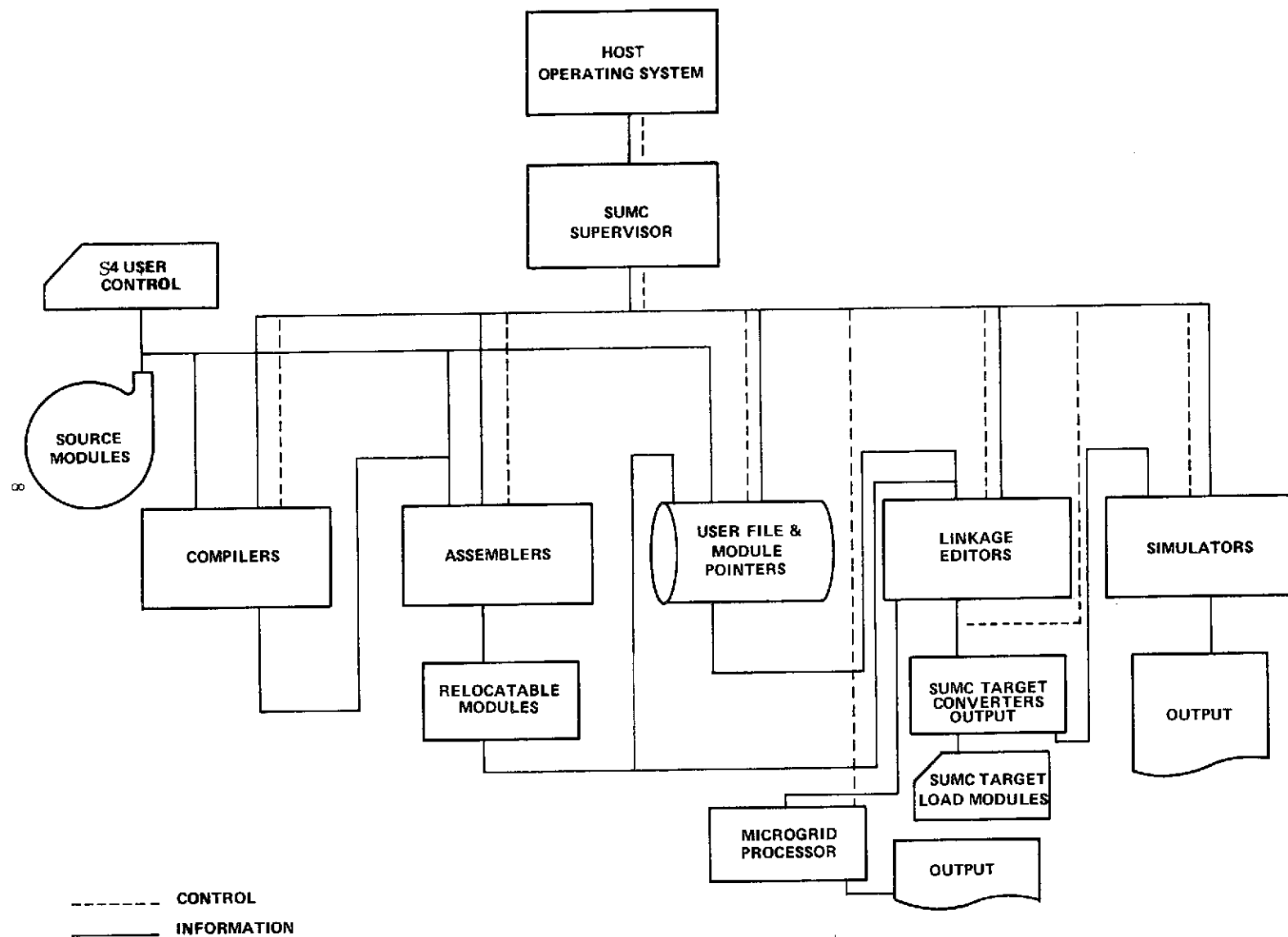


FIGURE 2. S4 SYSTEM PROCESSOR CONTROL AND DATA FLOWS

Instruction Assembler, which in turn produces relocatable object modules acceptable to the Linkage Editor. A load module which is generated by the Instruction Linkage Editor can then be used as input by the Simulator. A SUMC target module can be generated by the Target Output Converter from an object module or a load module. Microcode processing follows a slightly different procedure since the output requirements for microcode differ from the requirements for instruction level output. Output maps for microcode are needed to assist the microcode programmer's verification of the microcode. These differences are supported by the appropriate processors: (e.g., Microcode Assembler, Microcode Linkage Editor, and Microcode Grid Print).

B. S4 Supervisor

1. General. The S4 Resident Supervisor is made up of the components indicated in Table 2. The main purposes for this particular organization are:

- o Localize and minimize the problem of transportability from host to host by isolating most dependencies into a single processor; (these host dependencies include all assembly language programming).
- o Provide overlay control.
- o Provide a dynamic S4 system communications area.

The S4 Non-Resident Supervisor is a processor that executes under control of the S4 Resident Supervisor.

TABLE 2. S4 RESIDENT SUPERVISOR FUNCTIONS

DESCRIPTION	PURPOSE
Host Dependent Subroutines	Centralize all Assembly Language Host Dependent Subroutines.
S4 System FORTRAN Subroutines	Eliminate Redundancy and Centralize Routines Common to All Processors.
Common Data and Tables	Make System Data Available to all Processors.
Overlay Control	Control the Execution of Processors via Overlay.
S4 System Communication Area	Supervisor ↔ Processor Communications

The Non-Resident Supervisor's main purposes are:

- o Provide Control Card interpretation.
- o Communicate data to the Resident Supervisor for Processor Control.
- o Provide file maintenance for the applications programmer.
- o Provide utilities (e.g. listing, mapping, punching cards, etc.)

The present S4 design includes the following user functions:

- o Compilation - A FORTRAN Cross Compiler permits translation of ANSI FORTRAN IV language (with extensions) to an intermediate language for eventual processing and execution on a SUMC. The cross compiler will be discussed in detail in Section III.C .
- o Assembling - Cross Assemblers translate mnemonic statements to an intermediate language for eventual processing and execution on a SUMC. Assemblers provide capability to translate instruction level and microcode level commands. The assemblers will be discussed in Section III.D .
- o Simulation - Host independent simulation of a target computer is available at the instruction level. The simulator provides several dumps, traces and maps designed to expedite software debugging for the SUMC. A description of the simulation process is provided in Section III.F .
- o Linkage Editing - S4 gives the user the capability to link relocatable object modules to form load modules. Load modules can be a combination of library and user object modules. A description of the linking process is provided in Section III.E.

- o SUMC Target Output Converting - Reformats a load module or object module to make it acceptable to a particular target SUMC loader. A unique Target Output Converter will probably be required for each different SUMC target computer.
- o Line Editing - Facility for editing source modules on a line image basis. Replace, Insert, and Delete capabilities are provided.
- o File Mapping - Contents of files can be displayed by request of the user.
- o User Modification History Maintenance - A modification history is maintained as a part of the user files. This modification history is available to the user at any time.

2. Functional Description of the S4 Supervisor

a. Resident Supervisor. The Resident Supervisor is entered from the Host Operating system and executes the S4 System Initialization as indicated in Figure 3. When initialization is completed, the Resident Supervisor summons the Non-Resident Supervisor and passes control to it. The Non-Resident Supervisor reads and interprets an S4 control statement. A Processor Control Code is generated in the systems communication area. Control is then returned to the Resident Supervisor. The Resident Supervisor then summons the requisite processor overlay. Upon completion of execution of the requested Processor, control is returned to the Resident Supervisor. The Resident Supervisor then summons the Non-Resident Supervisor and passes control to it. This process continues until control is returned to the host operating system.

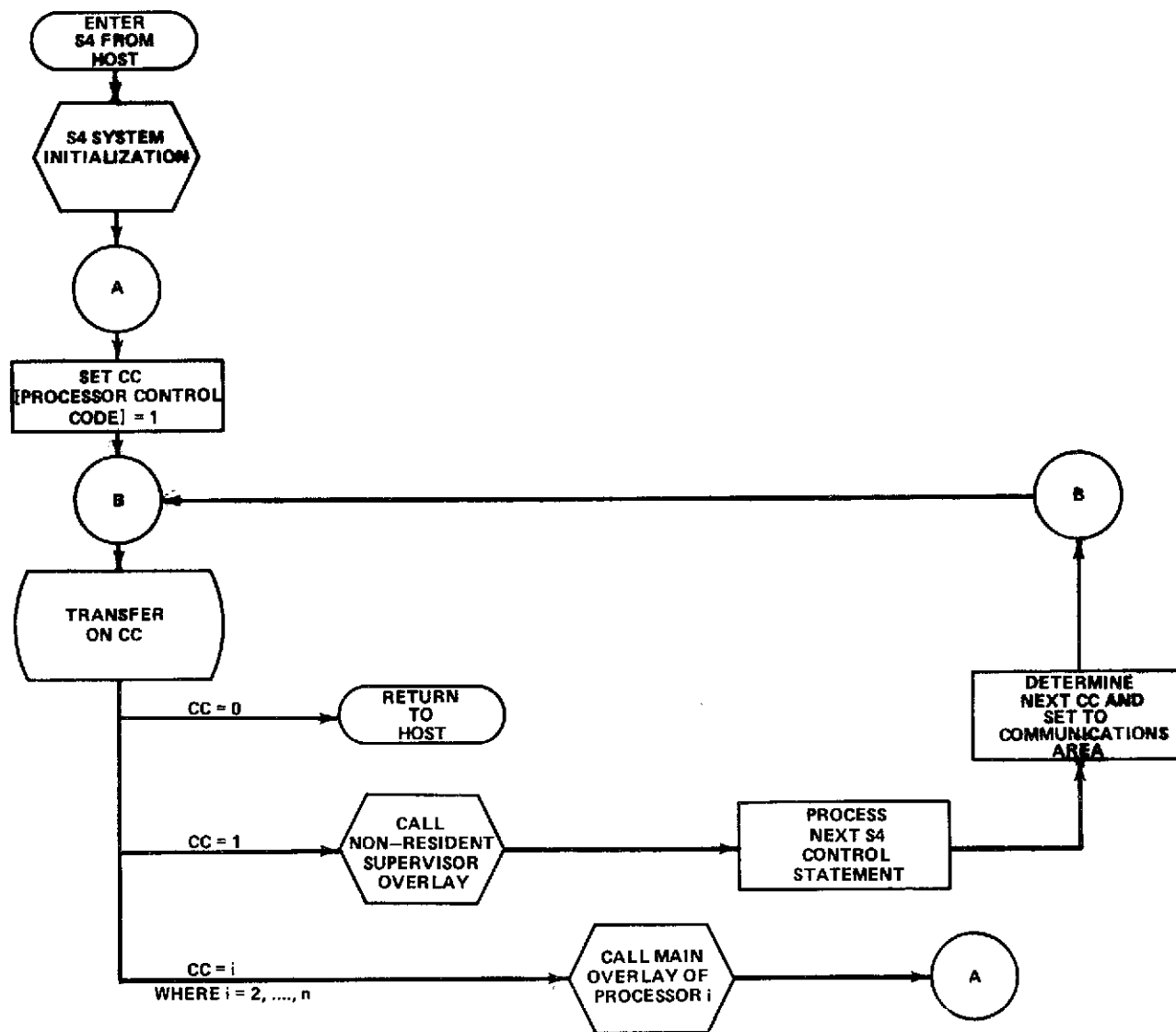


FIGURE 3. S4 SUPERVISOR OVERLAY CONTROL

b. Non-Resident Supervisor. The Non-Resident Supervisor is an open-ended processor which reads, interprets and executes or directs execution of the specified control statement function. A single control statement directs the Non-Resident Supervisor to perform one of the following:

- o Execute a processor (e.g. Assembler, FORTRAN IV Compiler, Linkage Editor, etc.).
- o Perform maintenance or mapping on a user file.

A user file is a collection of basic building blocks called user modules. A particular file or a module is identified by name. In addition, modules are identified by module type. A modification history is maintained at both the file and module levels. The user has freedom to construct an output file to include any modules that have been input or generated by the system. This freedom provides the user with the flexibility required in the generation and maintenance of programs for a SUMC computer.

These user facilities may be further described as follows:

- o User File Processing - Provides capability of inputting multiple user external files, thus creating an internal file. A user may output the internal file at any time during the job run. This will generate a new user external file containing all modules in the internal file at time of output. File mapping is available and provides listings of the module names and the characteristics of the modules in the internal file.
- o User Module Processing - Module utility functions are provided. Modules can be altered, listed or output to external devices. Alteration may include adding a module,

or line editing a module. A modification history of module alterations is maintained. The modification history can also be altered or deleted.

C. S4 FORTRAN Cross Compiler

1. General. The FORTRAN cross compiler is designed to translate input statements conforming to standards proposed by the X3.9 FORTRAN Group of the American National Standards Institute (ANSI).⁽³⁾ Minor extensions are planned to expand the compiler capability without impacting the transportability afforded by standardization.

2. Compiler Organization. The compiler organization is systematized as shown in Figure 4. The two major sections are called:

- o Functional elements.
- o Procedural elements.

The functional elements of the compiler are classified by the sequential control flow and are given in the large rectangular area of Figure 4 circumscribed by the dashed lines. Functional elements execute in several levels of control indicated by the simple tree structure.

In contrast, procedural elements are subprograms designed by collecting common operations present in the functional elements. A procedural element interface is constrained merely by its calling sequence. Within a procedural element the compiler designer is free to choose an algorithm without impacting design requirements of other elements.

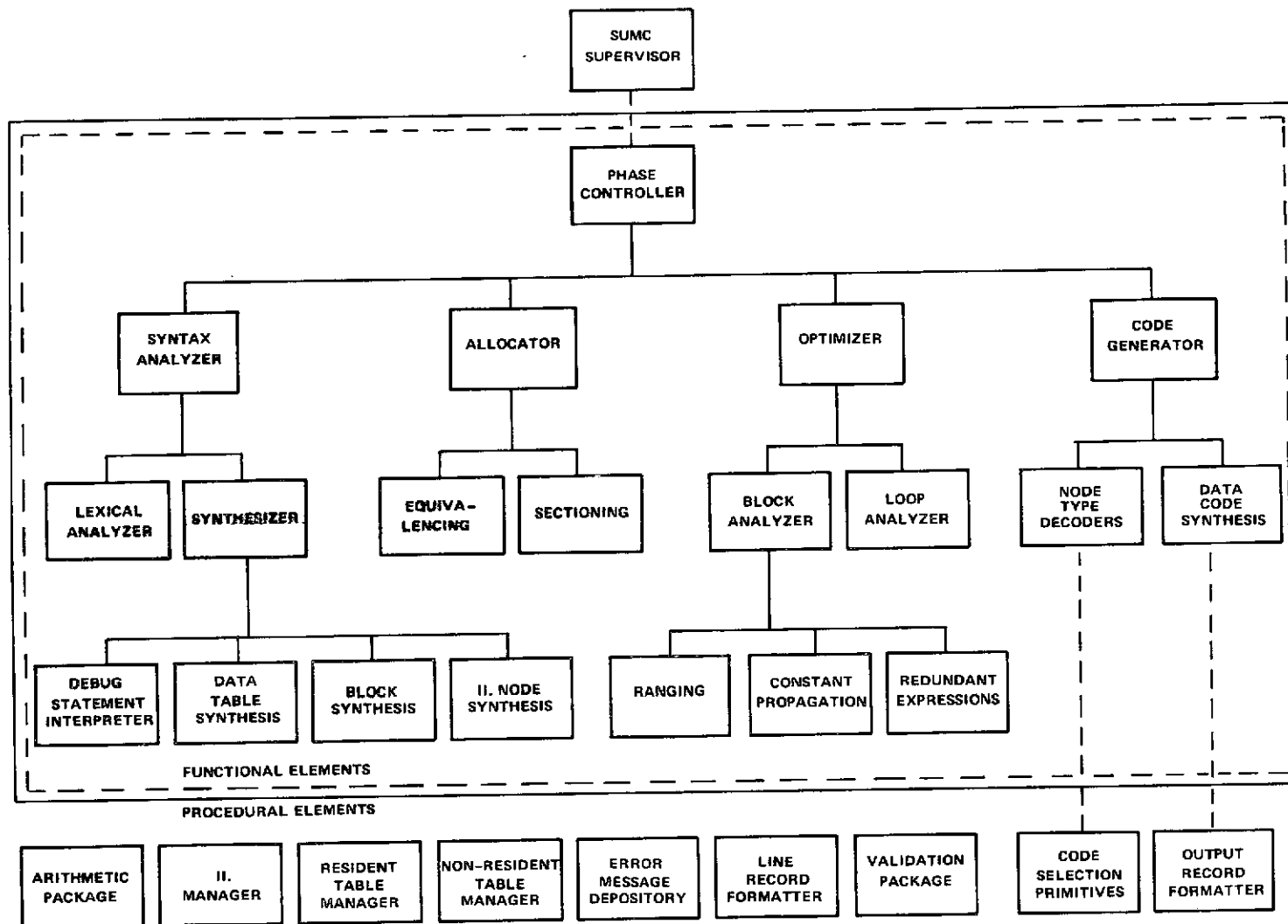


FIGURE 4. SUMC FORTRAN IV COMPILER DEPENDENCY CHART

The Phase Controller coordinates the execution of the functional elements which are divided into four phases and provides a single point interface for the Resident Supervisor. These phases are described below:

a. Phase 1. Comprises the source language statement transformations listed below:

(1) Lexical Analysis. The lexical analyzer performs the following functions:

- o Input source statements.
- o Print source statements.
- o Verify column alignment.
- o Determine statement type.
- o Combine symbols, numbers, and key words into single tokens for the syntax analyzer.

(2) Syntax Analysis. The syntax analyzer recognizes valid FORTRAN statements and displays error messages for the invalid statements.

(3) Synthesis. The synthesizer translates the source statements into an abstract syntax consisting of three elements:

- o Data Definition Tables - Derived partly from specification statements, and partly from variable usage in executable statements.
- o Control Structure - Division of the program into basic blocks, each with single entry and exit, delimited by control altering statements.
- o Compiler Intermediate Language - A division of expressions and control statements into more atomic elements (triples) that maintain the original semantics.

In addition, commands associated with the compiler validation package will be interpreted, the associated control tables built, and local optimization performed.

b. Phase 2. Allocation is a target computer dependent activity that includes:

- o Determining relative offsets for equivalenced data items.
- o Aligning data types and address boundaries commensurate with the requirements of the target computer instruction set.
- o Sectioning the data area into regions and assigning base registers.

c. Phase 3. Global optimizations performed in Phase 3 are types that are language dependent and target computer independent. An attempt has been made to isolate optimization to permit the analysis to be optional.

The optimizer will make one backward pass over the Intermediate Language (IL), analyzing basic blocks and loops. Analysis of blocks will entail:

- o Range of Variables and Definitions - Attaching to each simple variable reference the triple address of the next IL statement in which the variable appears, and the next redefinition address.
- o Constant Propagation - The evaluation of expressions containing variables with values known at compile time.
- o Redundant Expressions - The factoring of common arithmetic expressions occurring over consecutive statements.

- o Assignment Statement Permutation - Relocating assignment statements which may be encoded as part of a subsequent statement.

Although loops consist of several basic blocks, their predominant relationships can be discerned without extensive graph analysis. Therefore, a loop optimization is performed over multi-block regions. Two strategies will be applicable:

- o Invariant Expressions - Forming a loop prologue composed of expressions for which the operand values are not changed within the loop.
- o Operator Strength Reduction - Transposing multiplications involving inductive variables into a series of additions that can be performed optimally by incrementing index registers.

d. Phase 4. The code generation phase makes one forward pass over the Intermediate Language (IL), generating assembly language code for each triple not deleted in Phase 3. A subsequent pass over the allocation tables generates code representing the program's data structure.

Use of assembly language output rather than target machine code reduces the size of Phase 4 and more easily accommodates minor changes in the target computer instruction set.

The procedural elements of the FORTRAN Cross Compiler are noted in the lower portion of Figure 4 and are described in greater detail below:

a. Resident Table Manager. A number of compile time tables (symbols, constants, etc.) occupy a single storage area. The location and specific structure of these tables are hidden from other elements which may manipulate them only through the resident table manager. In this manner, overflow and storage reappportioning can be localized.

b. Non-Resident Table Manager. The Non-Resident Table Manager permits internal tables that overflow resident storage to be saved on mass storage for subsequent reference. This element localizes the interface to the supervisor's auxiliary storage service and enforces a uniform access procedure for the non-resident tables.

c. Intermediate Language (IL) Manager. The IL Manager consists of an additional internal table with associated manipulative functions. The IL manager translates higher level functions into the more primitive functions of the resident table manager.

d. Error Message Depository. A centralized routine constructs error messages for printing and appends the severity level.

e. Line Record Formatter. A number of different line printer record types are necessary. This element permits the combination of common functions, such as number base conversion and spacing.

- f. Validation Package (Debug). Facilities are available to:
- o Trace the syntactic decomposition of source statements.
 - o Trace selected routine calls within the compiler.
 - o Dump table contents in a symbolic format at selected execution points.

These tools enable the analysis of the compiler's operation on quite complex source programs early in the development phase. Otherwise the code generator would have to be operative before intermediate results could be verified for other than the most simple test case. The result of this earlier identification of logic errors greatly improves reliability of the final product. The Debug package should be permanently maintained with the compiler to ease future modifications resulting from changes in the host, target, or language.

g. Code Selection Primitives. This element is included among the procedural elements since it is desirable, on grounds of reduced effort required for modification, to separate it from the functional elements. The purpose of Code Selection Primitives is to receive calls from functional elements representing generic target computer operations and to translate these calls to the best of several instruction combinations.

h. Object Record Formatter. The purpose of this element is to construct object records in correct assembly language format and construct a source library element according to the supervisor standards.

i. Arithmetic Package. Since values originally obtained from the source statements enter into calculations (array offsets, equivalencing, constant propagation, etc.), it is convenient to save them in a binary format so they may be acted upon directly. The purpose of the Arithmetic Package is to perform the conversion of these values to a character form acceptable to the assembler.

D. S4 Cross Assemblers

1. Introduction. S4 Cross Assemblers process mnemonics representing SUMC instructions or predefined collections of computer instructions called macros. Cross assembler implies the execution of the S4 assembler on a host computer to generate machine instructions for a possibly different target computer.

The S4 Cross Assemblers are parameterized general assemblers. The S4 Cross Assemblers provide code generation capability for a wide range of SUMC target computers.

2. Instruction Assembler. This section identifies and summarizes the major characteristics of the S4 Instruction Assembler.

a. SUMC Word Length Independence. A fundamental design characteristic of the SUMC is a main memory word length that is expandable in four-bit increments. Consequently, the assembler word length is capable of specification in a corresponding manner.

b. SUMC Instruction Independence. The other fundamental design characteristics of the SUMC are:

- o Variable definition of main memory word subfields.
- o Target memory size specification.

c. SUMC Data Independence. The assembler permits definition of memory word length and decodable subfields for a SUMC target computer. The definable data formats are:

- o Byte, half-word, word and double word.
- o Fixed point, floating point and character.

d. SUMC Addressing Independence. The S4 Assembler permits definition of all types of addressing. These types are:

- o Immediate.
- o Direct.
- o Indirect.
- o Relative.
- o Indexed.

e. SUMC Character Translation. The SUMC target computer graphic character set is defined to the assembler. Any graphic that is not available on the host computer can be specified as a character constant.

f. System and Library Source Input. Assembler source input may be of two types:

- o User input.
- o Library input.

User input shall be considered the standard medium for assembler source statements. The assembler shall be capable of accepting and including in the input stream additional assembly statements from a source library.

g. Syntax Directed Parsing. A meta-linguistic, top-down, tree-structured parsing algorithm has been adopted in the assembler. It is anticipated that this approach makes the expansion of the assembler language definition possible.

h. Macro Processing. The Assembler has the capability to process and generate relocatable machine code for macro instructions.

i. Conditional Assembly. Often it is desirable to skip or repeat the assembly of selected source input statements. The assembler, therefore, includes conditional assembly instructions.

j. Assembler Expressions. Assembler expressions are convenient for the construction of program instructions and data. The assembler is capable of computing expressions containing arithmetic and logical operations.

k. External Symbols. It is often desired to subdivide a program into separately assembled subprograms and subroutines. In order to assemble these program sections separately and then combine them for execution, the Assembler accepts definitions of symbols that are external to (or local to) the current program section. Furthermore, the assembler communicates these external symbols to the linkage editor processor.

l. Program Relocation. Some programs (or sections of programs) require operation in fixed (absolute) locations in SUMC memory; others may execute from any (relocatable) location. The assembler is capable of differentiating relocatable instructions, addresses, and data from non-relocatable instructions, addresses, and data. Furthermore, the assembler communicates this program relocation information to the linkage editor processor.

m. Error Analysis. Errors of syntax or semantics may be present in the source input. Certain classes of assembly errors pertain to the improper designation or use of assembler instructions and are implicitly discernible to the assembler. Other classes of errors pertain to the construction and the intent of target defined machine operations, macro operations and data.

3. Microcode Assembler. The Microcode Assembler is an extension of the Instruction Assembler, with additional capabilities not required at the instruction level. This section identifies and summarizes these additional capabilities.

a. Microcode Assembler Definition. The SUMC microcode format does not lend itself to the separation of operations and operands. Therefore, there is a set of control fields, each dedicated to the utilization of a specific programmable hardware component. These control fields contain both operation and operand characteristics. There is an ordered and parallel action/ data flow within and between the hardware units addressed by these fields.

The microcode assembler language allows the specification of all control field settings. Instruction directives are defined which designate actual field settings for a selected combination of one or more control fields. A microcode assembler source statement is composed of one or more directives which collectively describe the operational function of a micro-instruction. This capability provides design flexibility through open-ended microcode assembler instruction sets.

b. Instruction Assembler Extensions. The Instruction Assembler extensions necessary for the Microcode Assembler are:

- o Rescan Source Operand - Provides the capability to set multiple fields based on an operand.
- o Field default values - Default values can be specified for those microcode fields not designated.
- o Instruction Read Only Memory (IROM) data structure - Allows the specification of the IROM data word structures.
- o Additional object output - The generated IROM data is output.

E. S4 Linkage Editors

1. Instruction Linkage Editor.

a. General. The Instruction Linkage Editor is a processor in the SUMC Support Software System. The linkage editor will receive inputs from the S4 cross assembler in the form of relocatable object modules. The linkage editor will interconnect these object modules to form a relocatable load module. The Instruction Linkage Editor provides an output data stream. The output of the linkage editor is referred to as a relocatable load module and is a generalized format.

Since SUMC target computer loader characteristics may vary from one application to the next, a target computer dependent conversion routine is required for processing a linkage editor relocatable load module. This routine called the "Target Output Converter" will reformat the generalized output of the linkage editor to make it acceptable to a particular target loader. The target output converter is dependent on the target SUMC Loader and usually will have to be recoded for each target SUMC Loader.

The Linkage Editor operation can be described in terms of its four principal processing operations:

- o Processing of Directives.
- o Load module formation.
- o Allocation of storage within a load module.
- o Post Process mapping.

b. Processing of Directives. The linkage editor is activated by an S4 LINK control statement. The S4 Supervisor processes this control statement and then passes control to the linkage editor. The S4 LINK control statement includes a load module name and load module entry point.

To construct a load module, the user must supply a set of linkage editor directives. The statements establish the name of at least one object module for the load module. These linkage editor interpreted directives include:

- o INCLUDE Directive - Used to collect object modules. The modules identified are included in the load module.
- o ABS Directive - Sets the linkage editor location counter.
- o END Directive - Used to indicate the end of a sequence of linkage edit directives.

c. Load Module Formation. A search of the user internal file is made for each object module that is named by the INCLUDE directives. If not found in the user internal file, then the system file is searched. Object modules not found in either file are undefined.

The entries in the internal and external label tables associated with each object module are collected in the load module global symbol table. A check is made to insure that all object module names in the load module global symbol table are resolved. If an object module name has not been resolved, then the same search procedure is followed for the implicitly defined modules as for those modules that were explicitly named on an INCLUDE directive. The checking and resolution procedure is followed until an attempt is made to define all implicitly defined modules. Those modules not resolved are noted as undefined.

d. Allocation of Storage. Object modules in a load module segment are processed in the order of their inclusion. Object modules named by INCLUDE directives are processed, and then those implicitly defined are processed. The first object module processed is assigned a base address. Subsequent modules are assigned an address immediately following the last address of the preceding module.

e. Post Process Mapping. The object module names and the characteristics of the modules are listed. The Global Symbol Table and its characteristics are also listed.

2. Microcode Linkage Editor.

a. General. The Microcode Linkage Editor accepts relocatable object modules which have been generated by the S4 microcode assembler and performs the following functions:

- o Links all external references to external definitions resolving all relocatable addresses.
- o Prints an absolute microcode assembler listing.
- o Generates an output microcode load module.

The Microcode Linkage Editor operation can be described in terms of the same principal processing functions outlined for the Instruction Linkage Editor:

- o Processing directives.
- o Load module formation.
- o Allocation of storage within a load module.
- o Post Process Mapping.

The last three functions of the above list have been described in paragraphs c., d., and e. of this chapter under Instruction Linkage Editor; the description applies to the microcode linkage editor.

b. Processing of Directives. The Microcode Linkage Editor is activated by an S4 MICLINK control statement. The S4 Supervisor processes this statement and passes control to the microcode linkage editor. The only information needed in the MICLINK control statement is a load module name.

To construct a load module, the user must supply a set of microcode linkage editor directives. These consist of:

- o ~~ABS~~ Directive - Controls the location of relocatable object modules as they are linked. The location counter initial value will default to zero.
- o INCLUDE Directive - Collects all the named object modules in the load module in the sequence specified.
- o END Directive - Terminates the linkage editor process.

F. S4 Instruction Simulator

1. General. The function of the Instruction Simulator is to interpretively execute a SUMC target load module. The load module will consist of SUMC executable code in the form of a target computer memory map. If the proper target computer architectural parameters have been provided to the simulator by the user, the program code will be interpretively executed by the simulator with all results being identical to those which would be achieved by the target SUMC computer.

The output or results which can be obtained from the instruction simulator include:

- o The normal execution output of the SUMC program under test.
- o Debug and verification information generated by the simulator diagnostics under control of user-supplied keys.
- o Simulation error information generated by the simulator in the event of an anomaly.

2. Functional Operation. The basic types of the simulation processor modular construction is shown in Figure 5. These modules and their functions are:

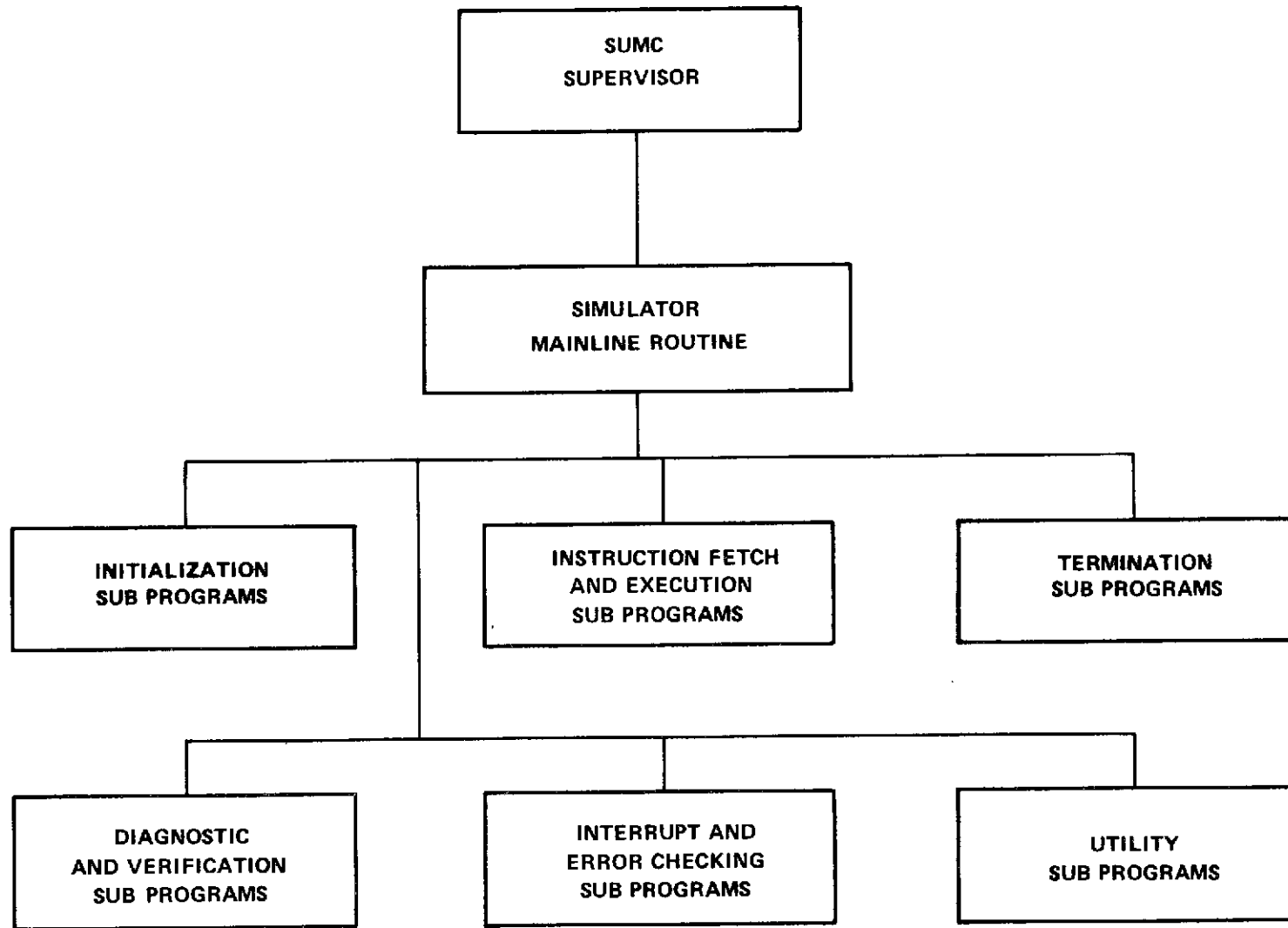


FIGURE 5. BASIC PROGRAM MODULES FOR SUMC INSTRUCTION SIMULATOR.

a. Simulator Mainline. The simulation process is carried out under control of the mainline routine in three primary phases:

- o Input of job description.
- o Interpretive execution of target program.
- o Termination of program.

b. Initialization. This set of program modules supervises the input of all external data and the initialization of all internal simulation parameters. The following data must be made available to the simulator prior to execution:

- o Values for target computer architectural parameters.
- o Target computer memory map.
- o Diagnostics keys and corresponding numerical data for diagnostic control.
- o Values for internal simulation parameters.

c. Instruction Fetch and Execute. The nucleus of the SUMC Interpretive Simulator is made up of two subroutines which simulate the instruction fetch and execute operations. The instruction fetch routine performs the following functions:

- o Validate instruction address.
- o Fetch instruction from simulated SUMC main storage.
- o Parse current instruction.
- o Validate all operand addresses and fetch required data.

The instruction execution routine performs the following functions:

- o Interpretively execute the current instruction.
- o Validate simulated execution.

- o Place results in appropriate register in target computer format.
- o Update statistics.

d. Interrupt and Error Checking. This set of program modules is responsible for simulation of the target computer interrupt detection, interrupt stacking and interrupt servicing. The simulator checks for interruptions after each instruction interpretation is completed.

e. Diagnostics and Verification. The SUMC interpretive simulator includes five types of diagnostic routines for simulated program verification:

- o SNAPSHOTS of selected target main memory locations.
- o TRACE of contents of key target registers.
- o DUMP of contents of scratch pad memory.
- o BLOCK DUMP of selected portion of target main memory.
- o FULL DUMP of target main memory.

User-supplied data is read by the program during the execution of the initialization routines and serves to activate or deactivate each of the above diagnostic aids. The diagnostics checking and processing are performed after each instruction fetch but before instruction execution.

f. Termination Programs. This set of program modules is used to control simulator operations following termination of target program interpretive execution. Whenever possible, the termination cause is identified and a table of program statistics is provided to the user.

g. Utility Programs. The simulator also contains a number of subroutines which perform generic operations and which may be used by a number of different simulator modules. The great majority of utility routines involve the execution of logical functions, bit manipulation operations or data conversion.

G. S4 Microcode Grid Print

1. General. The Microcode Grid Print processor enables the user to generate a print of an MROM (microcode read only memory) microcode load module. The user can specify the print format. It also permits the microcode load module mapping of all global symbols (both normal type and vector type) with their characteristics.

2. Processing of Directives. The microcode grid print is activated by an S4 MICGRID control statement. The S4 Supervisor processes this control statement and then passes control to the microcode grid print processor. The S4 MICGRID control statement includes the microcode load module name to be processed.

The user controls the grid print processor by the following directives:

- o PMAP Directive - Directs the mapping of global symbols in the microcode load module. All global symbols (both normal type and vector type) are mapped along with their characteristics.
- o HEAD Directive - Permits the user to specify up to 3 lines of headers at the top of page of the MROM grid print. If no header directives are specified, blank headers are printed.
- o FIELD Directive - Establishes a print field format for the MROM grid print. Bits from the MROM load module words can be concatenated in any desired sequence into a print field. A series of field directives (one for each print field) control the MROM print line format.
- o PROM Directive - Initiates the print of a microcode load module MROM grid. The Head and Field directives

specified previously control the format of the grid print.

The grid print can be specified as one of the following:

- Hexadecimal
 - Complemented hexadecimal
 - Binary
 - Complemented binary
- o END Directive - Indicates the end of microcode grid print.
Control is returned to the Supervisor.

SECTION IV. CONCLUSIONS AND RECOMMENDATIONS

Major processors of the SUMC System Support Software (S4) have been implemented in ANSI FORTRAN utilizing an XDS Sigma 5 as the host computer. Software has been processed by S4 generating object programs to execute on the SUMC target computer. Significant reduction of execution times for input/output bound programs has been accomplished using S4 input/output when compared with FORTRAN compiler I/O statements.

S4 is presently undergoing the modifications required for execution on an IBM 360-65. Changes are necessary in the areas of the S4 Supervisor data tables and in the host dependent subroutines.

The ease in transfer of S4 from the XDS Sigma 5 to the IBM 360-65 will be one measure of the success of the S4 design approach to host independence for a software support system.

The S4 system has produced the software development support required for a versatile spaceborne computer such as the SUMC. S4 allows the utilization of ground-based computers for generation and testing of operational flight software.

Future S4 expansion will include the addition of Processors. Those presently in the design stage are a FORTRAN Flow Charter, a Structured FORTRAN Program Translator, and a Systems Verification Language Compiler.

REFERENCES

1. Shuttle Computation System, E. Eastin, Contractor Rpt. SP-233-0252, Sperry Rand Corp., Huntsville, AL., June 1970.
2. National Computer Center, Standard FORTRAN Programming Manual, David and Charles, South Devon House, Devon, England, 1973.
3. Standard FORTRAN IV, ANSI-X3.9-1966, Engel, Frank, Jr., CBEMA, 179 Lewis Road, Belmont, MA. 02178.

APPROVAL

SYSTEM SUPPORT SOFTWARE FOR THE SPACE
ULTRARELIABLE MODULAR COMPUTER (SUMC)

by


T. E. Hill, G. C. Hintze, B. C. Hodges
Data Systems Laboratory

and

F. A. Austin, B. P. Buckles, R. T. Curran,
J. D. Lackey, R. E. Payne
Computer Sciences Corporation

The information in this report has been reviewed for security classification. Review of any information concerning Department of Defense or Atomic Energy Commission programs has been made by the MSFC Security Classification Officer. This report, in its entirety, has been determined to be unclassified.

This document has also been reviewed and approved for technical accuracy.



J. T. Powell, Director
Data Systems Laboratory

DISTRIBUTION

EF01/Mr. Powell/Dr. McDonough/Mr. Chambers
EF11/Mr. Swearingen/Mr. Frost
EF12/Dr. White
EF13/Mr. Eichelberger
EF14/Mr. Riley
EF15/Mr. Turner/Mr. Hodges/Mr. Hintze/Mr. Hill (75)
EF21/Mr. Hammers
EF31/Mr. Aichele
NA01/Mr. Lee/Mr. Grant
PF04/Mr. Brooksbank/Mr. Hall
NASA Hqts, Code MFC/Mr. Miller
AS 61 (2)
AS 61 L (8)
AT (6)
CC/Mr. Wofford
Scientific and Technical Information Facility (25)
P. O. Box 33
College Park, MD 20740